PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 28/96

PG-01 Regras e Recomendações para a Inclusão de Especificações no Código de Programas - Versão 1.00 -

Arndt von Staa
(Editor)

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, Nº 28/96
Editor: Carlos J. P. Lucena Setembro, 1996

PG-05 Regras e Recomendações para a Incusão de Especificações no Código de Programas - Versão 1.00 - *

Arndt von Staa
(Editor)

* Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brazil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

PG-05 Regras e recomendações para a inclusão de especificações no código de programas

Versão 1.02

A.v. Staa[1]
(Editor)
arndt@inf.puc-rio.br

Laboratório de Engenharia de Software
Departamento de Informática
Pontifícia Universidade Católica
22453-900 Rio de Janeiro
Brasil

PUC-RioInf.MCC28/96, Abril 1997[2]

Resumo: Neste documento estabelecemos regras e recomendação para inclusão de comentários de especificação no código de programas. Os comentários a serem incorporados são identificados através de marcadores classificando-os de acordo com o seu significado. A organização dos marcadores foi projetada de modo a permitir a geração automática de documentação técnica, onde a estrutura do hipertexto correspondente a esta documentação é extraída do próprio código, e o detalhe descritivo advém dos comentários marcados.

Palavras chave: comentários, especificação, geração automática de documentação, inteligibilidade, tipos de comentários, marcadores de seções de comentários, qualidade de documentação.

Abstract: In this document we establish rules and recommendations for including specification comments in program source code. Comments are classified according to their meaning. The meaning of every comment can be identified by means of the source text syntax and markup text contained in the comments. This assures the possibility to generate technical documentation directly from source code.

Keywords: automatic generation of technical documentation, comments, comment classes, documentation quality, markup language, specification, understandability.

---

[1] Trabalho apoiado por: CNPq, Bolsa de Pesquisador 300029/92-6, CENPES/Petrobrás, Itautec/Philco

[2] Uma versão anterior aparece como Monografia do Departamento de Informática: *PUC-RioInf.MCC 28/96*

Histórico de evolução

PG-05 Regras e recomendações para a inclusão de especificações no código de programas
Versão 1.02

Gestor: LES - Laboratório de Engenharia de Software, Departamento de Informática, PUC-Rio

Arquivo: Espec01
Editado: 15 maio, 1997
Impresso: 12 agosto, 2002

Documentos correlatos

PG-01 Regras e recomendações para a escolha de nomes de elementos em programas C e C++

PG-02 Regras e recomendações para o uso de constantes simbólicas em C e C++

PG-03 Regras e recomendações para a programação em C e C++

PG-04 Regras e recomendações específicas para a programação em C++

Versão V1.02

Editores: Arndt von Staa (PUC-Rio)

Status: Em uso

Data homologação: 01/jun/1997
Data entrada em vigor: 01/jun/1997

Data de início da próxima revisão 02/jan/1998

Descrição de evolução

- adaptação da norma de modo a viabilizar a extração automática de documentação técnica a partir de conjuntos de programas.
- Correção de erros de redação

Descrição da retroação

Versão V1.01

Editores: Arndt von Staa (PUC-Rio)

Status: Em uso

Data homologação: 01/mar/1996
Data entrada em vigor: 01/mar/1996

Data de início da próxima revisão 02/jan/1997

Descrição de evolução

- Correção de erros de redação

Créditos

Revisores

André Derraik	(TeCGraf PUC)	Versão 1.0
Christiano de Oliveira Braga	(LES PUC-Rio)	Versão 1.02
Geraldo Machado Costa	(LES PUC-Rio)	Versão 1.0
Lincoln Nobumiti Kanamori	(Itautec Philco)	Versão 1.0
Marcelo Jaccoud	(CENPES)	Versão 1.02
Pedro Alexandre O. Giovani	(Itautec Philco)	Versão 1.0
Pedro Jorge E. Hübscher	(LES PUC-Rio)	Versão 1.0
Renan Martins Baptista	(CENPES)	Versão 1.0
Rosa Maria Ramalho Correia	(Itautec Philco)	Versão 1.0

Apoio

CENPES Petrobrás
CNPq
Itautec Philco

Marcas registradas e nomes de produtos

MS-DOS e Windows são marcas registradas da Microsoft Corp

Sumário

Objetivo

O presente conjunto de regras e recomendações tem por objetivo estabelecer um padrão para a inclusão de comentários em programas C e C++, documentando:

- informações gerenciais relativas aos módulos e arquivos que compõem programas;
- especificações detalhadas dos diferentes componentes de um módulo.

Como resultado da adoção da presente norma espera-se que:

- os programas tenham um estilo de documentação consistente, uniforme e independente do autor;
- seja fácil entender corretamente um trecho de código sendo lido, independentemente de quem tenha gerado ou alterado este código;
- seja fácil modificar corretamente programas redigidos por outrem.

Os comentários de documentação definidos na presente norma são organizados de modo que se possa desenvolver ferramentas para a geração de documentação técnica diretamente a partir dos arquivos de código do programa. Esta documentação deverá estar em forma de hipertexto e deve ser, ao mesmo tempo, detalhada, completa, consistente, podendo ser dirigida para o programador cliente dos programas documentados, ou para o programador desenvolvedor ou mantenedor desses programas.

Motivação

Não existe dúvida sobre a necessidade de documentar adequadamente o código fonte de programa. Novas técnicas de programação, bem como a crescente complexidade dos sistemas modernos, e o desejo de aumentar a reusabilidade, tornaram a existência de documentação de importância fundamental no desenvolvimento dos sistemas. É reconhecido que, além de um bom projeto, é necessário um correto, completo, preciso e consistente registro das propriedades de cada componente. Sem este registro torna-se muito difícil a garantia de qualidade e o reúso de componentes.

Um bom estilo de programação, a escolha de bons nomes para os elementos de programas, e *a inclusão de detalhados comentários de especificação e projeto*, em muito facilitam a compreensão e manutenção dos programas. A documentação não deve se restringir apenas a explanar alguns aspectos dos algoritmos e das estrutura de dados utilizadas. Deve fornecer informações precisas quanto à especificação e ao projeto de cada um dos elementos. O formato e a composição destas informações deve ser uniforme. Em adição, o texto deve estar prontamente acessível aos desenvolvedores e mantenedores dos programas. Na falta de ferramentas mais eficazes, uma das formas de se atingir esta meta é incluir o texto de especificações no código sob a forma de comentários.

Além de informações técnicas, também são necessárias informações gerenciais, tais como, autores dos programas, datas de alteração e aprovação, histórico de versões, alterações realizadas e solicitações de alteração pendentes. Estas informações também devem ser incorporadas ao código dos módulos, a menos que se disponha de ferramentas mais eficazes.

No desenvolvimento de um software complexo, são produzidos diversos documentos de especificação, tais como especificações de requisitos, funcionais, da arquitetura e do projeto. É particularmente relevante ter-se à disposição as especificações detalhadas de cada um dos componentes de um módulo, tais como classes, funções, métodos, estruturas de dados e interfaces de usuário. A presente norma focaliza a documentação de módulos e de seus componentes, a documentação de nível de abstração maior, como por exemplo a especificação de sistemas, o projeto de arquitetura de sistemas, não são alvos da presente norma.

Mantendo-se o código e a especificação integrados em um único arquivo aumenta a disponibilidade da documentação. No entanto, pode tornar os arquivos de código fonte bastante extensos, dificultando, assim a localização da informação procurada. Em adição, a ausência de referências cruzadas aumenta a duplicação de informação, aumentando o risco de inconsistência, além de dificultar a localização da informação procurada. Uma forma de atenuar este problema é a utilização de *browsers* que, em um único hipertexto disponível *on-line*, permitem navegar e encontrar a informação procurada. Este hipertexto contém dados que advêm de diversos módulos, gerados a partir do arquivo único do módulo, de forma a manter a consistência entre a documentação e a implementação. Para facilitar a geração destes hipertextos a partir de um arquivo contendo código e

documentação, a presente norma identifica marcadores de seção a serem incluídos nos comentários. Estes marcadores serão utilizados por ferramentas geradoras de hipertexto para criar uma documentação mais rica do que a gerada somente com a análise sintática.

A forma de especificar adotada nesta norma é híbrida, no sentido de que se utilizam as declarações nas linguagens C ou C++ como parte da especificação. Por exemplo, ao invés de incluir um elemento de especificação definindo os métodos de uma classe, deixamos esta especificação para a declaração da composição da classe escrita em código C++. De forma semelhante, ao programar em C, utilizamos as declarações de protótipos de funções como instrumento para extrair a especificação da composição de módulos. Evita-se, assim, a redundância entre especificações e código, e a conseqüente dificuldade de assegurar a consistência das especificações e do código.

Um programa documentado de acordo com a presente norma conterá texto de código, comentários e marcadores embutidos nos comentários. Procurou-se seguir idéias semelhantes às encontradas em [Knuth 91], sem, no entanto, atingir o mesmo grau de detalhe. Desta forma espera-se assegurar um nível de pragmaticidade de uso, sem violar os conceitos básicos daquele artigo. Um texto criado de acordo com a presente norma é perfeitamente legível por um programador versado em C ou C++. Os marcadores adicionais são incluídos exclusivamente nos comentários e adicionam um mínimo de estrutura adicional. Portanto, um programa contendo estes marcadores pode ser compilado diretamente sem precisar passar por filtro algum. Por outro lado, pode-se passar o texto por processadores capazes de gerar arquivos em formato RTF (*Rich Text Format*), TEX ou HTML. Estes arquivos podem ser utilizados para gerar documentação formatada em papel, ou acessível via *browsers on-line*.

Em diversas ocasiões programas são formados por arquivos de código, arquivos de definição de constantes e arquivos de dados. Por exemplo, é comum que diálogos e menus sejam codificados através de arquivos de dados redigidos em linguagens destinadas a interfaces de usuários. A interface entre tais arquivos e o programa é usualmente realizada através de chaves numéricas passadas a funções de acesso aos dados contidos nos arquivos. Para assegurar a corretude[3] da interface, recomenda-se utilizar arquivos de definição de constantes simbólicas, ao invés de utilizar constantes literais no código. Estes arquivos de definição, se utilizados tanto pelos arquivos de dados como pelos programas-fonte, asseguram a corretude da referência a elementos de dados, mesmo que venha a ser realizada alguma evolução mais extensiva. Embora não sejam estabelecidas normas detalhadas para arquivos de dados e de definição, a presente norma apresenta regras e recomendações mínimas para tais arquivos quando utilizados para compor programas.

Definição da terminologia utilizada

É apresentado a seguir um pequeno glossário dos termos utilizados nesta norma.

módulo é uma unidade de compilação. Cada módulo deve implementar completamente uma única abstração bem definida. Exemplos de abstrações: o conjunto de funções (ou métodos) que implementam uma lista, um diálogo junto com todas as funções requeridas por este diálogo, ou o conjunto de operações tornadas disponíveis por uma aplicação. Evidentemente, um módulo – *módulo cliente* - pode lançar mão de um outro módulo – *módulo servidor* – para realizar as tarefas. Um módulo é implementado utilizando cinco classes de arquivos: módulo de especificação, módulo de documentação, módulo de declaração, módulo de implementação e módulo objeto, todos definidos a seguir.

módulo de documentação técnica[4] contém a documentação técnica necessária para que projetistas e programadores possam desenvolver, manter ou utilizar corretamente o módulo. O módulo de documentação aparece em duas modalidades: *documentação de interface* e *documentação de implementação*. A documentação de interface contém toda a informação necessária para que um programador possa corretamente utilizar módulos servidores em seus projetos. A documentação de interface está fortemente relacionada com o módulo de declaração a ser descrito

---

[3] *Corretude* (Correto + -tude) qualidade de estar correto. Preferimos utilizar corretude ao invés de *correção*, uma vez que este último é ambíguo, pois pode significar a qualidade de estar correto e a ação de tornar correta alguma coisa. O termo corretude tem sido largamente utilizado em publicações de teoria da computação.

[4] Quando utilizado na forma abreviada *módulo de documentação* estaremos sempre nos referindo ao módulo de documentação técnica.

mais adiante. A documentação de implementação complementa a documentação de interface, fornecendo toda a informação necessária para a correta implementação do módulo. O módulo de documentação não participa do processo de compilação, sendo estritamente um componente da documentação técnica. Do ponto de vista do processo de desenvolvimento, o módulo de documentação deve estar disponível *antes* de se desenvolver ou fazer uso de um determinado módulo.

módulo de documentação para usuário[5] contém texto descritivo da interface do usuário. O módulo do usuário visa o usuário da aplicação da qual o módulo faz parte. Tipicamente descreve aspectos operacionais de uso do módulo. O módulo de documentação do usuário pode ser utilizado para compor os arquivos de auxílio e os manuais destinados aos usuários da aplicação.

módulo de declaração[6] contém o código de interface do módulo. Este código de interface será utilizado ao compilar os módulos cliente, bem como ao compilar o correspondente módulo implementação. É importante frisar que existe um único módulo de declaração utilizado tanto pelos módulos cliente como pelo próprio módulo. Isto é necessário para assegurar, por compilação, a consistência entre os módulos compilados. Pode conter comandos de inclusão de arquivos definidos pelo compilador, de módulos de declaração de outros módulos – módulos servidores – e de arquivos de declarações de constantes.

módulo de implementação contém o código encapsulado do módulo. Contém ainda comandos de inclusão de arquivos do compilador, do *módulo de declaração próprio*, de módulos de declaração de outros módulos e de arquivos de declarações de constantes. O módulo de implementação é fornecido ao compilador para gerar o correspondente módulo objeto.

módulo objeto é o resultado da compilação de um módulo de implementação.

Conteúdo de uma especificação

Qualquer que seja o componente sendo desenvolvido, precisa-se saber:

- a finalidade deste componente;
- porque o componente é necessário;
- a interface e propriedades do componente;
- as condições de aceitação do componente. Estas são dadas por:
  - *requisitos*: condições, atributos, propriedades ou características necessárias para que a implementação do elemento seja aceitável. Em última análise, requisitos definem os critérios de aceitação que o componente sendo especificado precisa satisfazer por construção. Para poderem ser considerados bem formulados, deve ser possível verificar de forma inquestionável o atendimento de cada requisito.
  - *hipóteses*: condições, atributos, propriedades ou características assumidas como dadas antes de se construir o elemento sendo especificado. Quando se verificar que uma ou mais das hipóteses definidas não valem, é necessário reformular a especificação do componente. Para poderem ser consideradas bem formuladas, deve ser possível verificar de forma inquestionável a ocorrência de cada hipótese.
  - *restrições*: condições que restringem a liberdade de escolha de alternativas de construção do componente sendo especificado. Deve-se evitar a inclusão de restrições.

---

[5] Quando utilizado na forma abreviada *módulo do usuário* estaremos sempre nos referindo ao módulo de documentação para o usuário.

[6] Embora na terminologia estabelecida seja utilizado o termo *módulo de definição* (origem: Modula2) para designar as declarações de interface de um módulo, utilizaremos nesta norma o termo *módulo de declaração*, por ser mais comumente encontrado na literatura que trata de programação em C++.

Especificações definem o que um determinado componente deve realizar. Elas devem ser redigidas em uma linguagem precisa e inequívoca[7] para que diferentes leitores extraiam exatamente a mesma informação que o redator queria transmitir. É fundamental que todos os itens de uma especificação sejam objetivos.

Na presente norma as especificações são apresentadas sob a forma de fragmentos de texto contendo descrições, definições da interface, requisitos, hipóteses e restrições. As *descrições* destinam-se a dar uma visão geral da intenção, ou objetivo, do componente sendo especificado. Já a *definição* da interface, os requisitos, as hipóteses e as restrições devem ser redigidos em uma linguagem precisa. Idealmente deve ser utilizada uma linguagem formal. Nesta norma não adotaremos uma notação formal específica.

Exemplo de especificação de interface:

Assinatura[8]

EmpilharValor(pilha, valor) Þ pilha, condição de retorno[9]

Dados:

pilha	identifica a pilha na qual se adicionará o elemento
valor	referência para o valor a ser copiado para a pilha.

Resultado:

pilha	se não ocorreu erro:	conterá uma cópia do valor empilhado no topo pilha = [cópia valor] || pilha
	se ocorreu erro:	não terá sido alterada pilha = pilha
retorno	determina como a função concluiu a operação. Pode ser um de: OK Esgotou memória Pilha não existe.	

Requisitos

o tipo do valor deve poder ser qualquer, podendo, inclusive, variar de elemento para elemento de uma mesma pilha.

Hipóteses

espera-se que o valor a ser empilhado seja capaz de informar o seu tipo.

Interface física[10]:

```
tpCondRet Cpilha::EmpilharValor( const int   DimValor ,
                                 const void* pValor    )
   DimValor   - dimensão do valor em bytes
   pValor     - ponteiro para o valor
```

Exemplo de interface mal especificada:

```
tpCondRet Cpilha::EmpilharValor( const int   DimValor ,
                                 const void* pValor    )
   DimValor   - dimensão do valor em bytes
   pValor     - ponteiro para o valor
```

---

[7] *Inequívoco* – não ambíguo, fácil de classificar, fácil de observar, não dá margem a suspeita.

[8] A *assinatura de uma função* identifica e define todos os dados e estados recebidos e gerados ou alterados pelo programa. Os dados são apresentados na forma de uma lista entre parêntesis logo a seguir do nome da função, e os resultados são também uma lista de elementos após o "operador" *produz* (Þ). Na assinatura não é levado em consideração a forma de implementar a interface. Esta estará associada com o *cabeçalho da função.*

[9] Parâmetros podem ser simultaneamente de entrada e de saída. Neste caso, os parâmetros de entrada devem ser sublinhados para que se possa diferenciar entre o valor antes e após à execução da função. Por exemplo, quando observado como um todo, o valor da pilha antes, é diferente do valor após a ação de empilhar, quando bem sucedida.

[10] A *interface física* define o cabeçalho da função. Os elementos da assinatura que não aparecerem na definição do cabeçalho são implementados através de variáveis globais ou através de membros de classes.

- Faltou mencionar que existe uma pilha sendo modificada. Em C, com esta lista de parâmetros a pilha é global. Este fato deve estar claramente definido.
- Faltou dizer o significado dos valores possíveis de serem retornados.
- Faltou dizer como é realizado o empilhamento. Existe uma diferença significativa entre empilhar uma cópia de valor (por exemplo suponha que o valor seja um objeto) e empilhar uma referência (ponteiro) para o valor.
- Faltou dizer como é tratado o tipo dos valores a serem empilhados.
- Finalmente, faltou dizer o que acontecerá se ocorrer algum erro. É especialmente importante deixar claras todas as situações em que o resultado de uma função possa ser imprevisível.

Exemplo de requisito bem formulado:

- Diariamente, antes de iniciar o expediente será gerada, a partir da base da dados corporativa, uma base de dados de apoio à gestão contendo:
  - todos os itens de estoque em falta;
  - todos os itens de estoque com quantidade abaixo do nível mínimo;
  - todos os itens de estoque cujas vendas estão aceleradas com relação ao esperado;
  - todos os itens de estoque encomendados e cujo fornecimento está atrasado.

Exemplos de requisitos mal formulados

- Deverá ser assegurado um bom tempo de resposta. — O que é um bom tempo de resposta? Como medir?
- Na medida do possível o sistema deverá proteger-se contra erros de operação. — Como é que se sabe se foi ou não possível?
- Os procedimentos devem ser adequados ao usuário. — O que se entende por adequação ao usuário?

Exemplo de hipótese bem formulada

- O sistema de transmissão assegura a corretude da transmissão de todos os dados completamente propagados para o receptor.

Exemplo de hipótese mal formulada

- Existirão suficientes terminais para utilizar o sistema. — O que é suficiente?

Exemplos de restrições bem formuladas

- O sistema será desenvolvido em ANSI-C obedecendo às normas PG-01 a PG-08.

Exemplos de restrições mal formuladas

- o sistema deverá ser seguro. — Não é restrição, é requisito mal formulado. O que se entende por seguro?
- o sistema será desenvolvido em C. — Que C? A linguagem definida por algum compilador? Ou ANSI irrestrito? Ou ANSI-C restrito por normas internas? Por que escolher a linguagem de implementação ainda durante a especificação?

Regras gerais

Regra 1: Todas as seções de comentários gerenciais e de especificação têm o formato genérico a seguir:

```
//=================================================================
//$x1
//Texto da seção 1
//$x2
//Texto da seção 2
```

```
//...
//$xn
//Texto da seção n
//$.================================================================
```

onde:

xi	identifica a classe da seção de comentários. O Anexo 1 lista os identificadores padronizados.
Texto i	contém a especificação ou informação gerencial correspondente ao marcador xi.
$.	é o delimitador final da seção n.

Exceção 2: Serão explicitamente identificados os casos em que a identificação da seção deverá ser omitida.

Uma especificação detalhada é formada por diversas seções, por exemplo, especificações de classes, de funções, de tipos de dados. Cada seção contém um ou mais fragmentos de texto de especificação ou gerencial. Cada um destes fragmentos possui uma linha de cabeçalho que identifica esta seção. Esta linha possui um título e um marcador de identificação, xi. O título visa tornar a especificação inteligível, já o marcador permite que se desenvolva ferramentas capazes de gerar hipertextos contendo a documentação técnica dos programas.

A barra inicial da seção de documentação destina-se exclusivamente a facilitar a localização e a separação de comentários e código. Desta forma, além de produzir um texto organizado, facilita o seu manuseio.

Nos exemplos e nas definições que envolvam esquemas de código, será utilizado o padrão de comentários válido para C++. Para utilizar o padrão de comentários C, basta mudar o par de caracteres '`//`' para '`/*`' e terminar a última linha de um conjunto de comentários com '`*/`'. Para utilizar o padrão de comentários FORTRAN, basta substituir o par de caracteres '`//`' pelo caractere '`C`'. Por exemplo, seja seguinte a definição contida na norma:

```
//================================================================
//$AC <Gestor do arquivo>
//$AID
//   Projeto:           <nome do projeto>
//
//   Módulo de declaração:   <nome do arquivo>
//   Letras identificadoras: <letras>
//   Número identificador:   <número>
//
//   Versão:            V<vv.mmm>
//
//   Data de aprovação: dd/mm/aa
//
//   Autor(es):         <nome autor 1>
//                      <nome autor 2>
//
//$.================================================================
```

Ao utilizar C, o mesmo comentário terá o aspecto a seguir:

```
/*================================================================
*$AC <Gestor do arquivo>
*$AID
*
*     Projeto:           <nome do projeto>
*
*    <tipo arquivo>          <nome arquivo>
*    Letras identificadoras: <letras>
*    Número identificador:   <número>
```

```
*
*    Versão:            V<vv.mmm>
*
*    Data de aprovação: dd/mm/aa
*
*    Autor(es):         <nome autor 1>
*                       <nome autor 2>
*
*$.=============================================================*/
```

Finalmente, ao utilizar FORTRAN, o mesmo comentário terá o aspecto a seguir:

```
C=====================================================================
C$AC <Gestor do arquivo>
C$AID
C
C     Projeto:           <nome do projeto>
C
C    <tipo arquivo>          <nome arquivo>
C    Letras identificadoras: <letras>
C    Número identificador:   <número>
C
C    Versão:            V<vv.mmm>
C
C    Data de aprovação: dd/mm/aa
C
C    Autor(es):         <nome autor 1>
C                       <nome autor 2>
C
C$.=================================================================
```

Recom. 3: Comentários correspondentes a seções de especificação ou a informações gerenciais devem estar alinhados na primeira coluna.

Para tornar comentários de especificação mais visíveis, bem como para separar melhor as seções que compõem um programa, é conveniente que estejam bem delimitados no texto.

O texto a seguir ilustra a adoção dessas regras ao especificar uma função.

```
//=====================================================================
//$FC
//NOME:      Obter string formatado
//DATA:      3 de maio de 1995
//AUTOR(ES): Analista Um
//           Analista Dois
//
//$FD  Descrição:
//   A função cria um string formatado a partir de
//     - zero ou mais parâmetros, cada qual correspondendo a um valor a
//       ser formatado e incorporado ao string.
//     - um identificador de esquema de string
//
//   Um esquema de string é uma seqüência de caracteres entremeada de
//   campos. Cada campo inicia com o caractere de escape “&”. Cada campo
//   identifica o formato a utilizar e o índice do parâmetro a ser
//   inserido no lugar do campo.
//
//   A partir da identificação fornecida, a função copia o esquema
```

```
//   de string para o buffer, completa o esquema como os parâmetros
//   fornecidos devidamente formatados de acordo com as definições de
//   formatação contidos no esquema de string.
//
//   O esquema de string pode ser armazenado nos seguintes domínios:
//      programa
//      string table, Windows
//      arquivo de definição
//
//   A sintaxe de esquemas de string e a forma de preenchimento estão
//   descritos na classe: Esquema de string.
//
//$FE
//$FEP  Parâmetros de entrada recebidos
//   idMsg      identifica o esquema de string e o domínio onde está
//              armazenada
//              Formato: idString | idDominio
//                 idDominio pode ser um de:
//                    AX_DOM_MEM - o texto está em memória. Deve ser
//                                 utilizado quando Windows pode estar
//                                 em erro
//                    AX_DOM_STR - o texto está na string table do
//                                 programa. Deve ser usado quando não
//                                 existe arquivo de definição, ou
//                                 quando este estiver inoperante
//                    AX_DOM_DEF - o texto está no arquivo de definição.
//
//   dimBuffer  determina o tamanho máximo para a mensagem formatada
//   pszBuffer  ponteiro para o buffer onde será montada a mensagem.
//              Contém um valor indefinido ao entrar.
//   vtParm     vetor de parâmetros, para mais detalhes, ver a descrição
//              do tipo: Vetor de parâmetros
//$FEE  Dados globais encapsulados no objeto corrente
//   O objeto corrente deve referenciar um arquivo de definição
//   válido, contendo os esquemas de string buscados no domínio
//   arquivo de definição.
//
//$FS
//$FSV Valor retornado
//   AX_OK         a função formatou a mensagem corretamente
//   AX_ERR_BUFF   a função truncou a mensagem formatada para
//                 que coubesse no buffer
//   AX_ERR_STR    a função não encontrou o esquema de string no
//                 domínio definido. Neste caso o buffer conterá
//                 o string formatado correspondente ao seguinte
//                 esquema:
//                 “Mensagem <idStr> não existe no domínio <idDom>“
//
//$FSP Parâmetro retornado
//   pszBuffer     O buffer apontado conterá a mensagem montada.
//                 O valor do buffer será sempre um string zero válido
//$.=============================================================
```

Regras para informações gerenciais

Regra 4: Todos os arquivos iniciam com os seguintes comentários de informação gerencial:

1. identificação do arquivo
2. descrição do conteúdo

3. histórico de alterações

Cada arquivo utilizado deve conter uma identificação que deixe claro para o leitor a que projeto pertence, em que arquivo físico se encontra, autores e versão. O nome do arquivo é necessário para localizar arquivos físicos a partir de documentos impressos, ou a partir de textos contidos em *browsers*.

Identificação de arquivo

Regra 5: Todos os arquivos devem conter um comentário inicial identificando o arquivo. O esquema deste comentário é:

```
//=====================================================================
//$AC <gestor do arquivo>
//$AIx
//   Projeto:  <nome do projeto>
//   <tipo arquivo>          <nome arquivo>
//   Letras identificadoras: <letras>
//   Número identificador:   <número>
//
//   Versão:            <vv.mmm>
//
//   Data de aprovação: dd/mm/aa
//
//   Autor(es):         <nome autor 1>
//                      <nome autor 2>
//
//$ADx
//   Descrição do conteúdo do arquivo. Deve descrever o conteúdo do
//   arquivo e não fornecer uma especificação. As especificações estarão
//   vinculadas aos módulos, às classes, às funções, aos tipos e aos
//   atributos.
//$.=====================================================================
```

Neste esquema os campos têm o seguinte significado:

<x> identifica o tipo do arquivo. O valor de *x* será um de:

- **C** - tabela de constantes,
- **D** - declaração,
- **I** - implementação,
- **O** - documentação, utilizado no preâmbulo do arquivo contendo a documentação técnica ou do usuário.
- **T** - tabela de dados.

<gestor do arquivo> é a pessoa ou organização que possui o *copyright* do arquivo. O gestor é identificado pelo marcador `$AC`.

<nome do projeto> é o projeto em que foi contabilizada a última manutenção ou a criação.

<tipo do arquivo> é um de:

Módulo de declaração: contém o código de interface do módulo, ver norma *PG-03 Regras e recomendações para a programação em C e C++*.

Módulo de implementação: contém o código encapsulado do módulo.

Módulo do usuário: é uma extração dos módulos de definição e de implementação, acrescida de informação específica e que fornece informações para o *usuário* de uma aplicação que faça uso do módulo. Usualmente se destina a um gerador de arquivos de auxílio (*help*) ou a um formatador de textos. Freqüentemente este tipo de arquivo não possui uma estrutura pré-estabelecida.

Módulo de documentação: é uma extração dos módulos de declaração e de implementação utilizada para gerar a documentação técnica do módulo destinada aos engenheiros de software. Usualmente se destina a um gerador de hipertextos a serem explorados por um *browser* disponível ao engenheiro de software. Freqüentemente este tipo de arquivo não possui uma estrutura pré-estabelecida.

Tabela de constantes: é uma coletânea de declarações de constantes. ver norma *PG-02 Convenções para o uso de constantes simbólicas em C e C ++*.

Tabelas de dados: contém dados necessários para o sistema. São exemplos: definições de menus, de diálogos e de ícones.

<nome arquivo> é o nome e extensão do arquivo. A extensão deve estar em concordância com o tipo e a linguagem do arquivo. Este campo visa facilitar a localização do arquivo a partir de sua listagem, ou mesmo a partir do texto exibido por um *browser* de hipertexto.

<letras> são as letras identificadoras do domínio utilizadas para os elementos globais deste módulo. Ver norma: *PG-01 Regras e recomendações para a escolha de nomes de elementos em C e C++*.

<número> é o número base utilizado para criar as identificações (constantes) de elementos e recursos associados a este módulo, ver norma *PG-02 Convenções para o uso de constantes simbólicas em C e C++*.

versão identifica a última versão/modificação do módulo. É definida na forma *vv.mmm*, onde *vv* é o número da versão corrente, >= 1, e *mmm* é o número da modificação corrente, >= 0. *Versão corrente* é a versão do arquivo contendo o módulo. Uma evolução de versão corresponde a uma mudança de funcionalidade, ou de algoritmos. Uma modificação corresponde a uma correção ou a uma melhoria. Ao mudar a versão, a contagem de modificações retorna a 0.

<data aprovação> é a data em que a corrente versão/modificação do módulo foi aceita.

<autores> é a lista de autores que participaram da criação/alteração da corrente versão/modificação do módulo.

Regra 6: Os marcadores das seções de identificação dos arquivos são:

$AID para arquivos contendo módulos de declaração,
$AII para arquivos contendo módulos de implementação,
$AIC para arquivos contendo definição de constantes,
$AIT para arquivos contendo tabelas de dados,
$AIO para arquivos contendo módulos de documentação.

Regra 7: Os marcadores das seções de descrição dos arquivos são:

$ADD para arquivos contendo módulos de declaração,
$ADI para arquivos contendo módulos de implementação,
$ADT para arquivos contendo tabelas de dados,
$ADC para arquivos contendo definição de constantes,
$ADO para arquivos contendo módulos de documentação.

Exceção 8: Arquivos gerados por ferramentas adquiridas de algum fornecedor permanecem no formato definido por estas ferramentas.

Muitos sistemas utilizam tabelas de dados para conter informações de controle de comportamento. Por exemplo, é comum que aplicações Windows possuam arquivos de dados para definir diálogos, menus, e *string tables*. Estas tabelas fazem parte integrante do sistema e devem estar claramente identificadas.

Muitas das ferramentas utilizadas para gerar arquivos componentes de um sistema estabelecem formatos próprios. Pouco se ganha forçando o uso de outro formato. Por esta razão admite-se o uso do formato gerado por ferramentas, tal como mencionado na Exceção 8. No entanto, caso estes arquivos sejam editados a mão ou gerados com ferramentas desenvolvidas na própria instituição, o formato deverá obedecer o estabelecido.

Recom. 9: Todos os arquivos gerados por programa devem conter um comentário inicial identificando o arquivo. O esquema do comentário é semelhante ao esquema descrito na Regra 5, sendo que logo após a linha identificando o gestor do arquivo deverão constar linhas com o seguinte esquema:

```
//===============================================================
//
//   Gerado a partir de:     <nome arquivo>
//   ARQUIVO GERADO. Este arquivo não deve ser editado.
//
//===============================================================
```

Em muitas ocasiões são utilizadas ferramentas capazes de gerar arquivos. Por exemplo, é comum os arquivos contendo tabelas de constantes serem gerados a partir de arquivos de dados. Para assegurar consistência, estes arquivos não deverão ser editados. O cabeçalho dos arquivos gerados é quase idêntico ao cabeçalho do arquivo a partir do qual é gerado. Incluem-se os nomes do arquivo gerado e a advertência de que não deve ser editado.

Histórico de alterações

Regra 10: Cada alteração será registrada em um registro de histórico de alterações contido no arquivo. Este registro tem o esquema a seguir:

```
// ===============================================================
//$HA
//   Versão:       vv.mmm
//   Data:         dd/mm/aa
//   Autor(es):    lista de autores da alteração
//   Solicitações: lista de solicitações de alteração atendidas
//$HAD Descrição
//   Texto descrevendo a alteração realizada
//$.===============================================================
```

Recom. 11: Mantenha os registros de alteração em ordem decrescente de data.

Exceção 12: Não será criado registro de histórico relativo à criação inicial do arquivo.

Cada alteração aprovada realizada em um arquivo deve ser registrada. A seqüência destes registros deve ser mantida no arquivo, formando o histórico de alterações. O histórico de alterações deve ser mantido em ordem decrescente de data, ou seja, registros mais recentes devem preceder registros mais antigos. Note que seguindo esta recomendação o primeiro registro do histórico será igual ao cabeçalho do arquivo, exceto pela descrição. Tal como no cabeçalho do módulo, a data deve corresponder à data de aprovação da alteração. Desta forma somente serão registradas alterações efetivamente concluídas.

Alterações são realizadas para atender um ou mais relatórios de falha ou de solicitação de alteração. Os relatórios de falha e as solicitações de alteração devem ser registrados, e este registro possuirá um identificador. Cada alteração realizada deve relacionar os identificadores de todos os relatórios atendidos pela alteração.

A descrição da alteração deve descrever sucintamente a alteração realizada. No caso de históricos contidos em módulos, deve relatar os componentes do módulo que foram afetados – criados, alterados ou excluídos – pela alteração. Caso a alteração afete a funcionalidade do módulo ou componente, este fato deve ser anotado. A descrição do módulo ou componente será sempre relativa à versão mais atual.

Regras para arquivos contendo tabelas ou dados

Recom. 13: No caso de arquivos contendo tabelas de definição, descreva a natureza dos itens sendo declarados e o formato da tabela.

Recom. 14: No caso de arquivos contendo dados, descreva a sintaxe destes dados.

Recom. 15: Desenvolva os programas de modo que os arquivos de dados de configuração — por exemplo arquivos .INI e .DAT — utilizados pelo programa possam receber cabeçalhos com o formato acima.

Recom. 16: No caso de arquivos contendo dados de configuração utilizados pelo programa descreva a sintaxe da organização destes dados.

O exemplo a seguir exemplifica cabeçalho de um arquivo de dados:

```
//=================================================================
//$AIT
//   Projeto:  Ferramentas de suporte
//   Tabela de dados         ESQMSTR.TAB
//
//   Versão:            1.0
//
//   Data de aprovação: 03/01/97
//
//   Autor(es):         Autor 1
//
//$ADT Descrição, sintaxe
//   Cada linha do arquivo corresponde à definição de um esquema de
//   string, ou a um comentário.
//   Comentários são linhas em branco ou linhas iniciando com //.
//   Cada definição de esquema de string tem o formato:
//
//   <Nome C>  <Identificador>  <Domínio>  <Esquema de string>
//
//   <Nome C>  é o nome que será utilizado nos programas a guisa de
//             identificador simbólico do esquema de string
//   <Identificador>  é um número inteiro que serve de chave de acesso ao
//             esquema de string. O identificador é utilizado pela função
//             de acesso a esquemas de string. Para evitar colisão de
//             chaves, cada módulo deve definir uma faixa de valores
//             permitidos para esquemas de string (ver número
//             identificador do módulo).
//   <Domínio> determina o domínio de memória em tempo de execução que
//             conterá o esquema de string. Pode ser um de:
//               m  memória principal;
//               s  string table do Windows;
//               a  arquivo de definições
//   <Esquema de string> é uma constante string
//
//   Cada constante string pode conter 0 ou mais campos.
//   Os campos serão substituidos com dados do vetor de parâmetros
//   sendo formatados conforme consta na definição do campo
//   Para cada parâmetro o vetor de parâmetros define:
//       campo DimValor - número de bytes do parâmetro
//       campo pValor   - ponteiro para o valor, o tipo do valor é
//                        genérico
//
//   Campos tem o formato: %tn onde
//      %  é o caractere de escape sinalizando o início de um campo
//         Para inserir um caractere % no string é necessário fornecer
//         o par de caracteres \%
//      f  indica como o parâmetro será formatado.
//         Valores permitidos para f:
//           i  campo DimValor contém um valor inteiro a ser exibido
//              no string de saída
//           s  campo DimValor contém o tamanho de um string e
//              campo pValor contém o ponteiro para uma
//                 seqüência de DimValor bytes. Bytes com valor 0
```

```
//                 serão transformados para caractere branco.
//      n identifica o índice do parâmetro a utilizar.
//        0 é o primeiro parâmetro. Se n for maior do que o número de
//        parâmetros fornecidos, o campo será formatado "???n"
//
// A presente tabela será processada pelos programas
//   GERADEF.EXE  - comando de linha que cria uma tabela de definição a
//                  ser utilizada pelos compiladores C ou C++. As
//                  constantes simbólicas definidas são consistentes com a
//                  especificação da função ObterString.
//   GERATAB.EXE  - comando de linha que cria uma tabela de constantes a
//                  ser utilizada pelos compiladores C ou C++ para
//                  incorporar os esquemas de string residentes em memória
//                  aos programas
//   GERASTR.EXE  - comando de linha que cria uma string table compatível
//                  com Windows, contendo esquemas de string.
//   GERAARQ.EXE  - comando de linha que cria o arquivo de esquemas de
//                  string
//   ObterString  - função C capaz de recuperar e formatar um esquema de
//                  string identificado pelo par
//                  <domínio | identificador>.
//=================================================================
```

O exemplo acima é voltado para Windows. No entanto, como é fácil de observar, pode ser convertido para qualquer outro GUI (*Graphical User Interface*) sem maiores dificuldades.

Regras para módulos

Regra 17: Cada módulo deve conter uma descrição e, opcionalmente, um conjunto de requisitos, hipóteses e restrições. O esquema do cabeçalho de um módulo é:

```
//=================================================================
//$Mx
//NOME:  nome por extenso do módulo
//$MxD  Descrição
//   Texto de descrição. Deve identificar o objetivo do módulo.
//$MxU Modo de utilizar
//   Texto descrevendo como utilizar. Deve descrever como proceder para
//   incorporar o módulo a um programa e como o usuário interage com o
//   módulo.
//$MxD Descrição das estruturas de dados
//   No módulo de declaração, descreva as assertivas mais abstratas
//      das estruturas de dados definidas no módulo
//   No módulo de implementação, defina as assertivas estruturais
//      específicas
//$MxBIB Bibliografia
//   Relacione as referências bibliográficas utilizadas para
//   implementar o módulo.
//$MxR Requisitos assegurados pelo módulo
//   Texto dos requisitos
//$MxH Hipóteses assumidas pelo módulo
//   Texto das hipóteses
//$MxRS Restrições do módulo
//   Texto das restrições de implementação, de segurança ou outros
//$MO <nome do arquivo>
//$.=================================================================
```

onde **x** é um de:

D módulo de declaração

I módulo de implementação

Recom. 18: Inclua somente as seções para as quais exista algum texto. A seção descrição deve existir sempre.

Recom. 19: Inclua no módulo de declaração somente os elementos de especificação que precisam ser conhecidos pelos módulos cliente. Inclua os demais elementos no módulo de implementação.

O marcador $MO é utilizado para introduzir uma referência a um arquivo contendo algum módulo de documentação. Este marcador pode aparecer em qualquer lugar nesta seção.

Toda a documentação necessária para o correto uso de um módulo deve estar contida no módulo de declaração. Toda a informação complementar especificando como implementar deve estar contida no módulo de implementação. No caso de dúvida relativa a um dado item, verifique se é imprescindível conhecê-lo para fazer correto uso do módulo. Somente se a resposta for um *sim* bem justificado, inclua o item no módulo de declaração.

Deve-se evitar redundância de especificações. Cada especificação deve aparecer em um único lugar. Porém é comum que parte da especificação possa ser incluída no módulo, nas classes contidas no módulo, ou nas funções contidas no módulo ou nas classes. Para resolver esta ambigüidade de posição, incorpore a especificação no elemento **menos** abstrato. Se uma determinada especificação diz respeito a uma única função ou método, inclua essa especificação nesta função. Se diz respeito a diversos métodos de uma classe, inclua-a nesta classe. Somente inclua especificações no cabeçalho de um módulo em último caso. Ao programar em C estas situações serão freqüentes. Ao programar em C++ deveriam ser muito pouco freqüentes.

São exemplos de informações que devem estar presentes no cabeçalho de um módulo:

- O objetivo do módulo. Um módulo bem projetado implementa completamente uma única abstração bem definida. Esta abstração deve estar claramente identificada. Deve-se evitar espalhar porções de um mesmo módulo sobre diversos arquivos.
- Ao programar em C, abstrações de dados são implementadas por módulos. Assim, a descrição dos módulos que manipulem estruturas de dados devem descrever estas estruturas de um ponto de vista abrangente, sem detalhar a sua implementação. Já ao programar em C++ a descrição da estrutura de dados abstrata deve ser descrita na correspondente classe.
- O protocolo de uso das funções tornadas disponíveis pelo módulo. Pode ser necessário obedecer uma determinada ordem ao ativar as funções para que se mantenham íntegros estados de funcionamento do módulo.
- Referências a documentos de especificação.

A seguir ilustramos um cabeçalho de descrição de um módulo de declaração.

```
//==================================================================
//$MDC
//  Nome: Manipular esquemas de strings
//$MDD Descrição do módulo de declaração
//     Objetivos
//        - disponibilizar as funções de manipulação de esquemas de
//          strings
//        - possibilitar o uso de strings em vários idiomas sem requerer
//          uma versão de programa específica para cada idioma, contri-
//          buindo, assim, para o desenvolvimento de programas
//          multiidioma.
//        - permitir um tratamento eficiente e homogêneo de strings,
//          permitindo a eliminação de constantes string mantidas no
//          corpo do código dos módulos.
//
//$MDU Modo de utilizar
//     Ao desenvolver um programa:
//     - para cada módulo que utilize strings crie um arquivo de dados
//       definindo os correspondentes esquemas de strings. Veja a
//       definição contida em ESQMSTR.TAB.
```

```
//     - nos pontos de acesso a um string, constante ou composto, insira
//       uma chamada para a função ObterString, passando o identificador
//       do esquema de string e os parâmetros a serem introduzidos nos
//       campos contidos no esquema.
//
//$.=============================================================
```

Regras para classes

Regra 20: Utilize o esquema de comentários a seguir como cabeçalho de um classe:

```
//=================================================================
//$CC
//NOME:  nome por extenso da classe
//$CD Descrição da classe
//   Texto de descrição. Descreva o objetivo e as características
//   especiais da classe. O estilo do texto é tipicamente informal.
//$CT Parâmetros do modelo (template) da classe
//   Texto discriminando e explicando todos os parâmetros utilizados
//   no modelo da classe.
//$CMU Modo de utilizar a classe
//   Texto descrevendo como utilizar, em particular como utilizar
//   funções virtuais. Deve ser descrito também como incorporar esta
//   classe em um programa, e como o usuário interage com a classe.
//$CED Definição das estruturas de dados
//   No módulo de declaração, defina, no nível mais abstrato possível, as
//      assertivas estruturais das estruturas de dados disponibilizadas
//      pela classe
//   No módulo de implementação, defina as assertivas estruturais
//      relativas à forma de implementação das estruturas de dados.
//   O estilo do texto deve ser formal.
//$CBIB Bibliografia
//   Relacione as referências bibliográficas utilizadas para implementar a
//      classe. Se a referência pode ser de interesse de programadores que
//      farão uso desta classe, incorpore a referência no módulo de
//      definição. Em caso contrário, incorpore a referência no módulo de
//      implementação.
//$CR Requisitos assegurados pela classe
//   Texto dos requisitos
//$CH Hipóteses assumidas pela classe
//   Texto das hipóteses
//$CRS Restrições da classe
//   Texto das restrições de implementação, de segurança ou outros
//$CO <nome de arquivo>
//$.=============================================================
```

A descrição deve descrever a funcionalidade da classe e os serviços disponibilizados pela classe. Caso a classe requeira um protocolo de utilização específico, este deve estar descrito no módulo de declaração.

O marcador $CO é utilizado para introduzir uma referência a um arquivo contendo documentação sem estrutura pré-definida. Este marcador pode aparecer em qualquer lugar nesta seção.

Muitas classes implementam métodos que interagem através de estruturas de dados encapsuladas. Quando estas estruturas possuírem alguma complexidade é importante que as suas propriedades sejam registradas. No módulo de declaração são externadas as propriedades que o cliente da classe necessita conhecer. No módulo de implementação são registradas as propriedades específicas da implementação.

Por exemplo, considere uma tabela de símbolos. Uma definição da estrutura de dados contida em um módulo de declaração poderia ser:

- A tabela de símbolos possui 0 ou mais pares <*símbolo, identificação*>.
- Símbolos são seqüências de zero ou mais caracteres quaisquer.
- As identificações são números inteiros gerados pela classe e formam um conjunto denso.
- Cada par <*símbolo, identificação*> define uma relação um-para-um, ou seja:
  - para um determinado símbolo, caso conste da tabela, existe uma única identificação.
  - para uma determinada identificação, caso conste da tabela, existe um único símbolo.
- As identificações contidas em pares <*símbolo, identificação*> excluídos da tabela não serão reutilizadas.
- A dimensão máxima permitida para símbolos é informada pelo método `DimSimbolo` (em C a função `TS_ObterDimSimbolo`).
- O número máximo de símbolos permitidos na tabela é informado pelo método `NumMaxSimbolo` (em C a função `TS_ObterNumMaxSimbolos`).

Cabe salientar que o exemplo acima externa somente as propriedades que um cliente da classe precisa saber. Não é externada a forma de implementar a tabela. Ou seja, a descrição acima deve ser satisfeita por qualquer uma das possíveis implementações.

Uma determinada implementação deve ser descrita no módulo de implementação. Esta descrição complementa a informação contida na definição da estrutura de dados do módulo de declaração. Por exemplo:

- A tabela de símbolos utiliza um algoritmo de randomização com resolução de colisões através de listas de colisão.
- O vetor de randomização contém os ponteiros para as origens das listas de colisão.
- O índice de acesso ao vetor de randomização é calculado somando-se todos os caracteres do símbolo e obtendo o resto da divisão pelo tamanho do vetor.
- A lista de colisão é uma lista duplamente encadeada e que interrelaciona elementos contendo símbolos congruentes com um mesmo índice de randomização.
- Cada elemento da lista de colisão contém os valores: *símbolo*, *identificação*, além dos ponteiros na correspondente lista de colisão.
- A identificação do símbolo é formada por um par *<InxLista, Id>*, onde *InxLista* é o índice da lista de colisão que contém o símbolo e *Id* é o número único identificador do símbolo. Este esquema facilita localizar o símbolo quando se conhece o seu identificador.

No exemplo acima tomou-se o cuidado de não definir nomes de variáveis ou de constantes. Desta forma consegue-se manter a independência da linguagem de programação. Uma especificação completa da estrutura de dados constará então de:

- especificação da classe (inclusive métodos públicos), contida no módulo de declaração;
- especificação da classe (inclusive métodos privados), contida no módulo de implementação;
- especificação dos tipos e atributos utilizados na classe.

Caso a classe tenha sido implementada tendo por base alguma referência bibliográfica, referencie-a no fragmento de texto bibliografia. Além de dar créditos a quem originou a idéia, isto também ajuda um futuro mantenedor a entender o funcionamento da classe.

Finalmente, descreva na seção de hipóteses os pré-requisitos assumidos para a utilização da classe, tais como, a existência de bancos de dados específicos, as condições do ambiente de hardware ou do ambiente operacional, e descreva na seção requisitos os critérios de aceitação da classe. Cabe salientar que no módulo de declaração, requisitos e hipóteses são descritas sem que se considere uma determinada implementação. Já no módulo de implementação, devem ser fornecidos os requisitos, as hipóteses e as restrições específicas para uma determinada implementação.

Regras para funções e métodos

Regra 21: A descrição de uma função tem o seguinte esquema:

```
//=====================================================================
//$FC
//Nome: <nome por extenso da função>
//Assinatura: definição da assinatura completa ver seção 0.
```

```
//            Item opcional.
//$FD Descrição da função
//   Texto de descrição.
//$FBIB Bibliografia
//   Texto relacionando as referências bibliográficas utilizadas
//   para implementar a função.
//$FOV Sobrecarga (Overloading, somente para C++)
//   Texto descrevendo as diferenças do presente método com relação a
//   outros na estrutura de herança e que possuam o mesmo nome.
//$FE  Especificação dos dados e estados de entrada
//$FT Parâmetros do modelo (template) da função
//   Texto discriminando e explicando todos os parâmetros utilizados
//   no modelo (template) da função.
//$FEP Parâmetros recebidos
//      Texto discriminando e explicando todos os parâmetros da lista
//      de parâmetros. Deve especificar o nível de qualidade esperado.
//      Caso sejam utilizados elementos com tipos definidos pelo
//      programador, referencie estes tipos ao invés de copiar a sua
//      especificação para todos os elementos.
//$FEG Variáveis externas e públicas de entrada próprias do módulo
//      Texto discriminando e explicando
//      - cada variável global externa no presente módulo que contém a
//        função ou método e que será acessada antes de ser modificada.
//      - cada variável pública ou restrita de uma classe definida no
//        presente módulo, e que será acessada antes de ser modificada.
//$FEO Variáveis externas e públicas de outros módulos
//      Texto discriminando e explicando
//      - cada variável global externada por outro módulo e que será
//        acessada antes de ser modificada.
//      - cada variável pública ou restrita de uma classe definida em
//        outro módulo, e que será acessada antes de ser modificada.
//           Cabe salientar que variáveis externas, públicas e restritas
//           devem ser utilizadas com parcimônia. Ao invés delas deve-se
//           dar preferência a funções de acesso.
//$FEA Arquivos de entrada ou atualização
//      Texto discriminando cada arquivo a serem lidos ou atualizados pela
//      função.
//$FES Estados ao entrar
//      Texto discriminando os estados que o equipamento e as estruturas
//      de dados devem satisfazer antes de ativar a função.
//$FEE Variáveis globais encapsuladas
//      Texto contido no módulo de implementação discriminando
//      - cada variável global encapsulada no presente módulo, e que
//        será acessada antes de ser modificada.
//      - cada variável privada de classe, e que será acessada antes de
//        ser modificada.
//$FS  Especificação dos dados e estados ao retornar
//$FSV Valor retornado
//      Texto descrevendo a natureza do valor retornado, discriminando
//      valores especiais. Por exemplo, caso a função retorne uma condição
//      de retorno, cada um dos possíveis valores deverá ser identificado
//      e descrito.
//$FSP Parâmetros alterados
//      Texto discriminando e explicando os parâmetros da lista de
//      parâmetros cujo valor poderá alterado. Devem ser discriminados
//      os espaços de dados alterados recebidos por ponteiro.
//$FSG Variáveis externas e públicas de saída, próprias do módulo
//      Texto discriminando
```

```
//      - cada variável global externa do presente módulo e que poderá
//        ser alterada pela função
//      - cada variável pública ou restrita de classe definida no presente
//        módulo e que poderá ser alterada pelo método
//$FSO Variáveis externas e públicas de outros módulos
//      Texto discriminando
//      - cada variável externa de outro módulo e que poderá ser alterada
//        pela função
//      - cada variável pública ou restrita de classe definida em outro
//        módulo e que poderá ser alterada pelo método
//$FSA Arquivos de saída
//      Texto discriminando cada arquivo no qual a função poderá gravar ou
//      alterar registros
//$FSS Estados ao sair
//      Texto discriminando os estados que equipamento e estruturas
//      de dados satisfarão ao terminar a execução da função
//$FSE Variáveis globais encapsuladas e privadas
//      Texto contido no módulo de implementação discriminando
//      - cada variável global encapsulada do módulo que poderá
//        ser alterada.
//      - cada variável privada da classe, que poderá ser alterada pelo
//        método.
//$FIU Especificação da interface com usuário
//$FDG  Diálogos realizados com o usuário
//      - Texto identificando ou descrevendo os diálogos ativados ou
//        processados pela função.
//$FMN  Menus e comandos manipulados pela função
//      - Texto identificando ou descrevendo as opções de menu tratadas
//        pela função.
//$FMSG  Mensagens exibidas para o usuário
//        Texto identificando ou descrevendo as mensagens de processamento
//        que poderão ser exibidas, discriminando as causas que levam à
//        sua exibição. O texto deve identificar cada um dos botões e
//        explicar o efeito se for apertado.
//$FFB   Mensagens de feedback de estado
//        Texto descrevendo as mensagens de estado que poderão ser
//        externadas pela função
//$FHLP  Auxílio
//        Texto identificando ou descrevendo as mensagens de auxílio
//        contextual que podem ser ativadas ao executar ou em conseqüência
//        da execução da função.
//$FR Requisitos assegurados pela função
//        Texto de requisitos adicionais, tais como tempos de resposta
//        esperados, limitações de memória a serem obedecidas
//$FH Hipóteses assumidas pela função
//        Texto de hipóteses adicionais.
//$FRS Restrições da função
//        Texto de restrições de implementação, de segurança, etc.
//$.=============================================================
```

Regra 22: Os cabeçalhos das funções externadas e dos métodos aparecem tanto no módulo de declaração (protótipo) como no módulo de implementação. Não aparecem no módulo de declaração os cabeçalhos de funções encapsuladas.

Regra 23: Cada instituição poderá definir explicitamente a posição relativa do cabeçalho ou protótipo e correspondente comentários de especificação. Não existindo uma definição explicita associada a esta norma, deve-se redigir os comentários de especificação imediatamente antes dos correspondentes protótipos ou cabeçalhos.

Na maioria dos livros texto e na maioria dos programas existentes, os comentários de especificação antecedem o correspondente cabeçalho ou protótipo. Sugere-se, então, adotar preferencialmente esta organização como regra. Por outro lado, algumas instituições já vêm adotando sistematicamente a alternativa de primeiro redigir o cabeçalho ou protótipo e, depois, o respectivo comentário de especificação. Para evitar a necessidade de se reorganizar grandes quantidades de código legado, optou-se por uma solução em que instituições podem definir um complemento a esta norma sempre que não seguirem a organização *default*.
O item assinatura é opcional, uma vez que a descrição detalhada de todos os dados de entrada e dos resultados corresponde, em última análise, à assinatura, tendo muito mais riqueza de detalhe. No entanto a assinatura pode ser interessante uma vez que ela apresenta a interface completa no mais elevado nível de abstração. Também aqui a instituição deverá estabelecer se a assinatura deverá ou não figurar no comentário. A opção *default* é não figurar.

Regra 24: O módulo de declaração deverá conter as especificações de interface das funções externadas e dos métodos públicos ou restritos, fornecendo as informações necessárias para que se possa ativar e utilizar[11] corretamente a função ou método.

Regra 25: O módulo de implementação deverá conter as especificações adicionais de implementação das funções externadas e dos métodos públicos ou restritos, necessárias para corretamente implementar ou compreender a implementação.

Regra 26: O módulo de implementação deverá conter as especificações de funções encapsuladas e de métodos privados, fornecendo tanto as informações necessárias para que se possa ativar e utilizar corretamente a função ou método, bem como as necessárias para corretamente implementar ou compreender a implementação.

Regras para tipos

Regra 27: A descrição de um tipo tem o formato e o conteúdo a seguir:

```
//=============================================================
//$TC
//NOME   <nome por extenso do tipo>
//$TD Descrição do tipo
//      Texto de descrição. A descrição deve se ater ao tipo como um todo,
//      deixando para a descrição em cada atributo do tipo (campo) o
//      detalhamento do campo.
//$TBIB Bibliografia
//      Relaciona as referências bibliográficas utilizadas
//$TAE Assertivas estruturais
//      Texto definindo precisamente o tipo, ver item Descrição da
//      classe na seção Regras para classes. A definição deve observar o
//      tipo como um todo. Por exemplo, se o tipo define um elemento de
//      lista, a assertiva estrutural deve definir a lei de formação e as
//      regras de corretude desta lista.
//$TR Requisitos assegurados pelo tipo
//      Texto dos requisitos adicionais
//$TH Hipóteses assumidas pelo tipo
//      Texto das hipóteses adicionais
//$TRS Restrições do tipo
//      Texto das restrições de implementação, de segurança, etc.
//$TEV Valores possíveis para tipo enumeração.
```

[11] Para evitar ambigüidade de terminologia, recomenda-se o uso do termo *ativar* sempre que se descreve propriedades de uma função do ponto de vista do programador. O termo *utilizar* deve ser usado para somente para descrever como o usuário interage com a função.

```
//$.============================================================
```

Regra 28: Tipos públicos devem ser especificados no módulo de declaração. A especificação deverá preceder imediatamente a declaração do tipo.

Regra 29: Tipos privados devem ser especificados no módulo de implementação. A especificação deverá preceder imediatamente a declaração do tipo.

Regra 30: Em enumerações utilize o marcador $TEV para definir o significado de cada valor da enumeração.

Regras para atributos

Regra 31: As definições e declarações dos atributos (dados, variáveis) devem preceder as definições dos métodos.

Regra 32: Os atributos devem ser definidos em ordem alfabética dentro de sua classe de visibilidade.

Regra 33: Cada atributo definido em uma `class` ou em uma declaração `struct` ou `union` deve ser especificado segundo o esquema a seguir. A especificação será posicionada na linha imediatamente a seguir da declaração e alinhado em margem esquerda 3 caracteres para a direita da margem esquerda do nome do atributo.

```
//$DC
//NOME   <nome por extenso do atributo>
//$DD Descrição do atributo
//   Texto de descrição. Deve descrever o atributo e a sua medida.
//   Exemplos de medidas:
//      - taxa de transmissão em caracteres por segundo;
//      - velocidade em m/s;
//      - nome de pessoa;
//      - endereço completo.
//$DV Critérios de validade do atributo
//   Texto descrevendo os valores válidos, ou como é controlada a
//   validade do atributo.
//$DR Requisitos assegurados pelo atributo
//   Texto dos requisitos adicionais
//$DH Hipóteses assumidas pelo atributo
//   Texto das hipóteses adicionais
//$DRS Restrições do atributo
//   Texto das restrições de implementação, de segurança, etc.
//================================================================
```

Regras para a redação de requisitos, hipóteses e restrições

Recom. 34: Assegure que todos os requisitos, hipóteses e restrições do componente sendo especificado estão explicitamente definidos.

Recom. 35: Assegure que todos os requisitos, hipóteses e restrições do componente sendo especificado são necessários para corretamente implementá-lo.

Recom. 36: Assegure que todos os requisitos, hipóteses e restrições do componente sendo especificado são suficientes para corretamente implementá-lo.

Recom. 37: Assegure que todos os requisitos, hipóteses e restrições do componente sendo especificado são consistentes entre si e com as demais especificações.

Recom. 38: Assegure que todos os requisitos, hipóteses e restrições do componente sendo especificado são precisos, exatos, rigorosos, concisos[12] e inequívocos.

[12] Está se tornando comum na literatura de garantia de qualidade utilizar o termo *precisão* com o sentido de limitação de erros computacionais, inclusive os decorrentes dos métodos de cálculo numérico. O termo *exatidão* tem sido

Recom. 39: Assegure que todos os requisitos e restrições do componente sendo especificado possam ser verificados quanto à sua realização.

Recom. 40: Assegure que todos os requisitos, hipóteses e restrições do componente sendo especificado são redigidos de forma compreensível para os diversos leitores.

Recom. 41: Assegure que todos os requisitos, hipóteses e restrições sejam aceitáveis do ponto de vista do serviço a ser prestado pelo componente sendo especificado.

Recom. 42: Assegure que todos os requisitos, hipóteses e restrições do componente sendo especificado são viáveis.

Recom. 43: Assegure que todos os requisitos do componente sendo especificado são independentes da tecnologia e plataforma utilizada para a sua implementação.

Recom. 44: Para cada conceito utilize somente um mesmo nome. Para cada nome tenha somente um conceito.

Um nome neste contexto pode ser um nome de código, um nome simples ou um nome composto por diversas palavras. Esta recomendação segue o espírito das regras e recomendações contidas na norma *PG-01 Regras e recomendações para a escolha de nomes de elementos em C e C++*, ampliado para o uso de nomes contidos em textos. O ideal é manter um glossário de nomes e os respectivos significados e assegurar que cada nome que ocorra no texto corresponda a um dos nomes neste glossário e que o significado subentendido no texto é exatamente igual ao significado definido no glossário. Por exemplo, o que é um *item de estoque*? Um objeto físico que se encontra no estoque? Um tipo de objeto que se pode adicionar ou retirar do estoque? Um número de objetos de um determinado tipo encomendado e registrado em um pedido de compra? Cabe salientar que o uso descuidado de palavras é a principal fonte para falhas de entendimento de especificações.

Como conseqüência desta recomendação devem ser evitados sinônimos. Devem ser evitados pronomes, a menos que o contexto deixe perfeitamente claro o valor que representam. Note que pronomes são variáveis que assumem um valor definido pelo contexto em que são empregados.

Recom. 45 Utilize somente palavras que existam e que exprimam conceitos perceptíveis:

verbos:	ações cujos resultados são visíveis ou mensuráveis
substantivos:	é possível observar o objeto ou medir a sua existência
adjetivos:	definem propriedades perceptíveis dos substantivos
advérbios:	definem propriedades perceptíveis de adjetivos ou verbos

Evite palavras do tipo “basicamente”, “a filosofia do componente é...”. Evite aportuguesamentos de termos em inglês, prefira “excluir” a “deletar”, “marcar” a “setar”. Utilize palavras com um único significado, utilize “maior” para denotar “maior do que”, mas nunca para denotar “maior ou igual”. Uma boa regra é verificar se cada palavra fará falta se retirada da frase. Se não fizer, retire-a. Evite adjetivação inútil.

Recom. 46: Use frases curtas e sintaticamente corretas. Ao encontrar uma frase longa:

1. Quebre-a em frases sucessivas, separadas por pontuação. Assegure que a quebra faça sentido e que esteja gramaticalmente correta.
2. Ou particione a frase em uma lista de itens sucessivos. Cada idéia estará num item.

Uma frase curta tem tipicamente 16 ou menos palavras. No entanto, não leve esta recomendação ao pé da letra. A capacidade de comunicação do texto é mais importante do que o tamanho da frase.

Recom. 47: Explicite condições e assegure que estejam completas.

---

utilizado no sentido de conformidade com o mundo real. No Aurélio é insinuada a sinonímia destes termos. Os termos *rigoroso* e *conciso* aparecem no Aurélio como propriedades de ser exato ou preciso. A enumeração explícita de cada um desses termos permite que se use cada um com uma acepção exata.

Não diga “maior do que 5”, quando 5 fizer parte do conjunto. Neste caso diga: “5 ou mais”. Quando uma condição for composta, itemize e endente seguindo as mesmas recomendações que as encontradas na norma *PG-03 Regras e recomendação para programação em C e C++*.

Recom. 48: Procure utilizar verbos na voz ativa.

Recom. 49: Evite o uso de frases aninhadas e apostos.

Recom. 50: Evite o uso de negações duplas e de negações iniciando uma frase. Procure utilizar um estilo afirmativo.

Anexo 1 Marcadores de seção

$.	fim da seção de comentários marcados
$AC	identificação do gestor do arquivo
$ADC	descrição do arquivo tabela de constantes
$ADD	descrição do arquivo módulo de declaração
$ADI	descrição do arquivo módulo de implementação
$ADO	descrição do arquivo contendo documentação
$ADT	descrição do arquivo de dados
$AIC	identificação do arquivo tabela de constantes
$AID	identificação do arquivo módulo de declaração
$AII	identificação do arquivo módulo de implementação
$AIO	identificação do arquivo documentação
$AIT	identificação do arquivo de dados
$AV	versão e modificação corrente do arquivo
$CBIB	referências bibliográficas da classe
$CC	cabeçalho da classe
$CD	descrição da classe
$CED	descrição das estruturas de dados implementadas pela classe
$CH	descrição das hipóteses assumidas pela classe
$CMU	descrição do modo de utilizar da classe
$CO	referência a arquivo contendo documentação da classe
$CR	descrição dos requisitos assegurados pela classe
$CRS	descrição das restrições da classe
$CT	descrição dos parâmetros de modelo (*template*) de classe
$DC	cabeçalho do atributo (dado, variável)
$DD	descrição do atributo
$DH	descrição das hipóteses do atributo
$DR	descrição dos requisitos do atributo
$DRS	descrição das restrições do atributo
$DV	descrição dos critérios de validade do atributo
$FBIB	referências bibliográficas da função
$FC	cabeçalho da função ou método
$FD	descrição da função
$FDG	descrição dos diálogos utilizados ou processados pela função
$FE	subseção dados de entrada da função
$FEA	descrição dos arquivos contendo dados de entrada ou atualizados
$FEE	descrição dos dados globais encapsulados utilizados antes de alterar
$FEG	descrição dos dados e tipos globais próprios utilizados antes de alterar

$FEO	descrição dos dados e tipos globais de classes ou módulos servidores utilizados antes de alterar
$FEP	descrição dos parâmetros recebidos
$FES	descrição dos estados da estação requeridos ao entrar
$FFB	descrição das mensagens de *feedback* de estado[13] geradas pela função
$FH	descrição das hipóteses da função
$FHLP	descrição do auxílio contextual associado à função
$FIU	subseção da interface de usuário
$FMN	descrição dos menus e comandos de menu processados pela função
$FMSG	descrição das mensagens emitidas pela função
$FOV	descrição das propriedades de sobrecarga (*overloading*)
$FR	descrição dos requisitos da função
$FRS	descrição das restrições da função
$FS	subseção dos resultados produzidos
$FSA	descrição das alterações realizadas em arquivos de saída ou atualizados
$FSE	descrição das alterações realizadas nas variáveis encapsuladas
$FSG	descrição das alterações efetuadas nos dados globais próprios
$FSO	descrição das alterações realizadas nas variáveis globais de classes ou módulos servidores
$FSP	descrição das alterações retornadas via parâmetros
$FSS	descrição das alterações realizadas nos estados da estação ao sair da função ou método
$FSV	descrição do valor retornado pela função
$FT	descrição dos parâmetros de um modelo (*template*) de função
$HA	cabeçalho do item de histórico de alteração
$HAD	descrição do item de histórico de alteração
$MBIB	referências bibliográficas vinculadas ao módulo
$MDC	cabeçalho do módulo de declaração
$MDD	descrição do módulo de declaração
$MDH	hipóteses assumidas pelo módulo de declaração
$MDR	requisitos assegurados pelo módulo de declaração
$MDRS	restrições do módulo de declaração
$MDU	descrição do modo de utilizar do módulo
$MIC	cabeçalho do módulo de implementação
$MID	descrição do módulo de implementação
$MIH	hipóteses assumidas pelo módulo de implementação
$MIR	requisitos assegurados pelo módulo de implementação
$MIRS	restrições do módulo de implementação

[13] Mensagens de *feedback* de estado são mensagens que informam o estado corrente do processamento ao usuário. Em diversos aplicativos estas mensagens aparecem no rodapé da janela.

$MO	referência para arquivo contendo documentação
$TAE	descrição das assertivas estruturais do tipo
$TBIB	referências bibliográficas do tipo
$TC	cabeçalho do tipo
$TD	descrição do tipo
$TEV	descrição de um item de enumeração
$TH	descrição das hipóteses do tipo
$TR	descrição dos requisitos do tipo
$TRS	descrição das restrições do tipo

Bibliografia

[Braga 95] Braga, C.O.; Staa, A.v.; *Ferramentas para a Geração Automática de Documentação*; Relatório Técnico, PUC-RioInfMCC 14/95; Departamento de Informática, PUC-Rio; 1995

[Brown 89] Brown, H.; "Standards for Structured Documents"; *The Computer Journal* vol. 36 no. 6; 1989; pags. 505-514

[Cowan 94] Cowan, D.D.; Germán, D.M.; Lucena, C.J.P.; Staa, A.v; "Enhancing code for readability and comprehension using SGML"; *IEEE International Conference on Software Maintenance ICSM'94*; Victoria British Columbia; September 19-23, 1994;

[Cullens 97] Cullens, C. et alii; *Usando Visual C++*; Rio de Janeiro; Campus; 1997

[Ellis 93] Ellis, M.; Stroustrup, B.; *C++ Manual de Referência Comentado*; Rio de Janeiro; Campus; 1993

[Figueiredo 94] Figueiredo, L.H.; Celles, W.; *Documentação de Software no TeCGraf*; Relatório Técnico, TeCGraf; Departamento de Informática, PUC-Rio; 1994

[Knuth 91] Knuth, D.E.; *Literate Programming*; Center for the Study of Language and Information; 1991